International Journal of Robotics Research and Development (IJRRD)
ISSN(P): 2250-1592; ISSN(E): 2278-9421
Vol. 11, Issue 2, Dec 2021, 45–54
© TJPRC Pvt. Ltd.

VISION-BASED OBSTACLE AVOIDANCE ON QUADCOPTER

KASHIF KHURSHID NOORI[1], MANISH KUMAR MISHRA[2] & MD. ZAFARYAB ABDULLAH[3]

[1]*Robotics Engineer, Jamia Millia Islamia, New Delhi, India*

[2]*A.I Engineer, Jamia Millia Islamia, New Delhi, India*

[3]*Robotics Engineer, Jamia Millia Islamia, New Delhi, India*

ABSTRACT

Obstacle avoidance is an integral part of the Autonomous Navigation of Unmanned Vehicles, whether it be Aerial, Ground, or Underwater vehicles in indoor or outdoor environments. In a broader sense, it comprises obstacle detection and avoidance strategies. With the availability of multiple types of range-finding sensors like Lidar, Radar, or Ultrasonic sensors, a vision sensor is considered the best candidate for obstacle avoidance owing to its low cost, weight, and provision of enriching information about the environment. Motivation for vision-based avoidance owes its basis to the fact that humans, birds, insects use only vision for obstacle avoidance and navigation.

This work proposes a vision-based obstacle detection, a planner for obstacle avoidance, and way-point navigation on Quadcopter using a low-cost Xbox Kinect RGB-D camera. Simulation of the work is examined on the Gazebo Simulator with the Robot Operating System (ROS)-Indigo framework, Pixhawk as Flight Controller Unit (FCU).

KEYWORDS: UAVs, ROS, Kinect Sensor, Obstacle Avoidance & Way-Point Navigation

Received: Aug 26, 2021; **Accepted**: Sep 16, 2021; **Published**: Sep 28, 2021; **Paper Id.**: IJRRDDEC20214

1. INTRODUCTION

For any autonomous system, obstacle avoidance is a key research problem. In recent years, we have witnessed much application of the quadcopter in the field of agriculture, surveying, search and rescue or logistics, and integration of autonomous capabilities will significantly increase quadcopter's operations in these sectors. We could use various sensors to solve obstacle avoidance like Stefan Hrabar [1]uses a 3D occupancy map of an environment using stereo vision and a path planner for generating collision-free trajectory, B. A Kumar [2] and Kwag [3] uses Radar or Lidar by A. Bachrach [4]. However, Lidar and stereo vision are computationally complex [5], also quadcopters have limited payload capacity; it's inefficient to carry typical radar [6].

In this paper, we are using the XBOX Kinect sensor [7] for obstacle detection and distance estimation. We have also worked on Behaviour-based robotics [8] to develop avoidance strategies.

2. QUADCOPTER IN GAZEBO

A. Simulation Software

We have used Gazebo version 7 with ROS-Indigo on Ubuntu 14.04 LTS OS. A quadcopter Gazebo model, Erle-Copter which uses RotorS [9]simulation plugins to simulate Quadcopter kinematics and sensors, Ardupilot SITL Gazebo plugin to establish communication between simulated quadcopter and Gazebo. Furthermore, to get every data on the ROS platform, we have used the MAVROS package.

B. Reference Frame

We need to deal with these reference frames and transformation between them while working with Quadcopter, following figure [Figure: 1] shows various reference frames attached to the Quadcopter.

- World Frame: The odometry frame of the Quadcopter. We get position $\vec{D}$, and orientation of Quadcopter in this frame over respective ROS topics.
- Quad-Body Frame: It is fixed to the quadcopter's body and has the same initial axis orientation as of the World frame.
- Camera Frame: It's attached to the Xbox-Kinect camera of the quadcopter.
- Quad-Velocity frame: Quadcopter in Gazebo Simulator accepts Velocity in this frame.

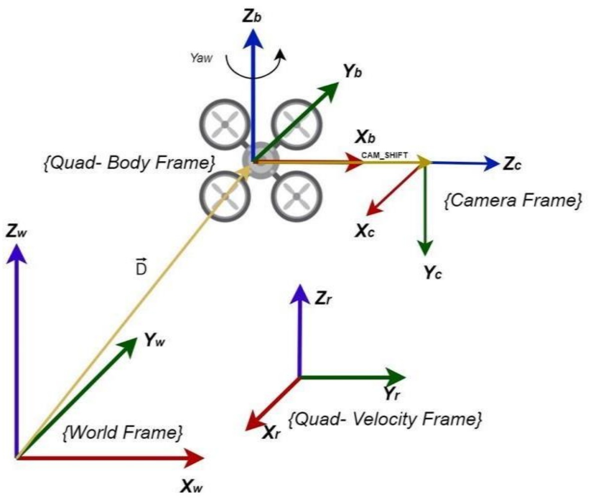


Figure 1: Reference Frames.

We also need to tackle two transformations:

- Forward Heading Velocity in Quad-Body frame to Quad-Velocity frame:

Assuming V_l be Linear velocity and yaw angle be θ in Quad-Body frame. V_x, V_y are velocity components in the X-Y direction of the Quad-Velocity frame.

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} -V_l * sin(\theta) \\ V_l * cos(\theta) \end{bmatrix} \tag{1}$$

- Position in Camera frame to World frame:

Assuming $\vec{D}$ be the position of the quadcopter in the world frame, and p be the distance of the camera from the Quad-body frame.

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} sin(\theta) & 0 & cos(\theta) & p * cos(\theta) + D_x \\ -cos(\theta) & 0 & sin(\theta) & p * sin(\theta) + D_y \\ 0 & -1 & 0 & D_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

C. Dynamics of Quadcopter

The integration of Ardupilot SITL helps to access the Guided flight mode of Arducopter. The guided mode enables the copter to receive velocity commands in an inertial or body frame of reference by a computer, in our case a ROS node will send it. As we could directly control the quad's velocity, it drastically simplifies the quadcopter's dynamics. Assuming U be input velocity in the World frame. Thus its dynamics will be:

$$\underline{\dot{x} = U_x \quad \dot{y} = U_y \quad \dot{z} = U_z}$$

Here *x,y,z* are *Position* & are the velocity of the Quadcopter in the World frame of reference.

3. OBSTACLE DETECTION

A. Microsoft Kinect Sensor in Gazebo

The Kinect sensor released by Microsoft Corporation is a low-cost motion-sensing device to enhance the gaming experience. It has three sensors: RGB camera, an IR projector, and an IR camera. Gazebo simulator provides Openni Kinect Plugin which simulates Kinect sensor and publishes both Image and Depth data on the same topics as the corresponding ROS drivers for the Microsoft Kinect. It publishes RGB image data in ROS Image datatype and Depth data in ROS PointCloud2 data type.

B. Object Detection

We are detecting objects with the help of Computer vision using OpenCV with Python 2.7. Gazebo Simulator publishes Image data from the simulated Xbox-Kinect sensor over

ROS topic *"/front_cam/kinect/rgb/image_raw"*.

We have used OpenCV bridge, a ROS package to convert Image data received in ROS msg format to Image format supported by OpenCV. The following figures illustrate the steps required in our algorithm to detect obstacles.

- Canny Edge: Canny Edge detector [10] is one of the most used detectors to detect edges in a given frame.

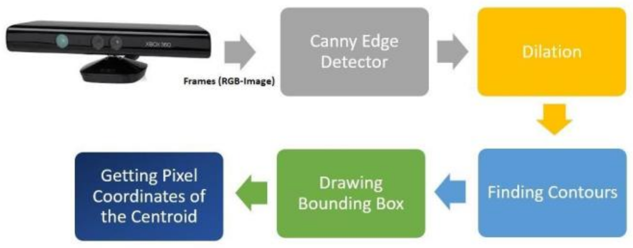


Figure 2: Obstacle Detection Algorithm from Xbox-Kinect's RGB Camera.

It uses a gaussian filter to remove noise in the image. The algorithm computes gradients and applies threshold on gradient values. Also, it discards the weak edges that are not connected to strong ones.

- Dilation: It is a method to increase the focused pixel area using a kernel of different sizes. After applying the canny edge there are still some edges left that need to be amplified, for that purpose we have used the dilation method. It can be achieved by first performing erosion which removes white noise (also shrinks) followed by dilation.

- Finding Contours: to find contours in the image after dilation we have used cv2 function to find all contours in the image, also computed the hierarchy between contours (small contours inside large contours), which further gave us the dimension of the final contours present in the image.
- Drawing Bounding Box: After getting the dimensions of the contours using cv2, we have used another cv2 function to draw a box that bounds the contours.
- Getting pixel of the centroid: To get the depth of the obstacle using point cloud data we need to find a single point of the obstacle for which we took the centroid of the box.

C. Pixel to X_c Coordinate

The Gazebo Simulator publishes Depth data in form of

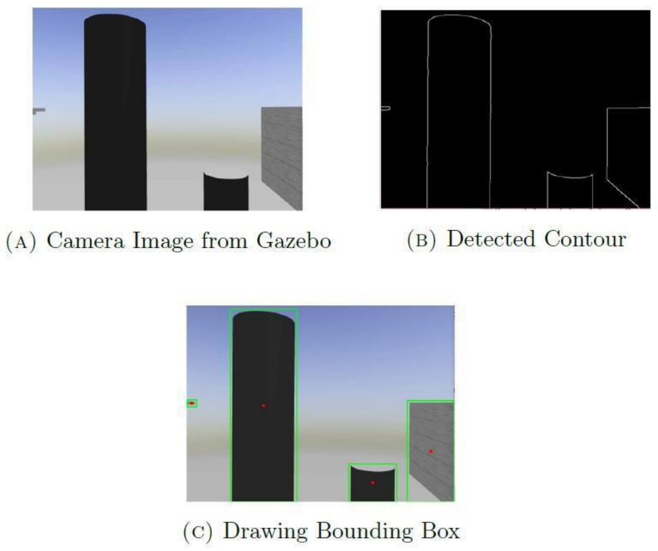
”sensor_msgs/PointCloud2” datatype[11], which stores X-Y-Z depth data in an 1-D array. To extract pixel coordinates, it has ”row step” and ”point step” attributes which helps to find index value from which coordinates data begin in depth data array. We have followed following steps to extract X-Y-Z of pixel value *u, v*:

- array pos $\leftarrow u * point\_step + v * row\ step$
- X Byte data will be from array_pos to array_pos+4
- Y Byte data will be from array_pos+4 to array_pos+8
- Z Byte data will be from array_pos+8 to array_pos+12
- Convert all these data from Byte to Float numbers

4. QUADCOPTER NAVIGATION

A. Behaviours of Quadcopter

The environment present around us is fundamentally unknown and always changing with the progression of time.



(A) Camera Image from Gazebo (B) Detected Contour

(C) Drawing Bounding Box

Figure 3: Finding Contour and Drawing Bounding Box Around Obstacles.

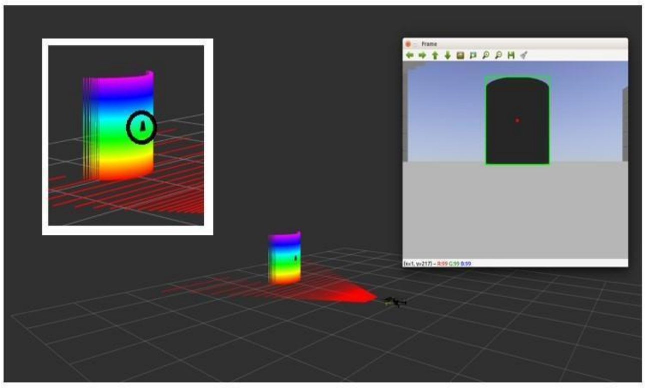


Figure 4: Visualizing Pixel in RGB Image and Corresponding XYZ Coordinate of that Pixel in Rviz.

So, it does not make sense to equip a robot with a pre-defined plan for its functioning in this environment. It would be better if we could design different behaviors or control of the robot and switch among these behaviors in response to environmental changes, which summarizes the key idea behind Behavior-based robotics. We have defined three behaviors for navigating our robot autonomously without slamming into obstacles in an unknown indoor/outdoor environment. We have designed all these behaviors for a Point Robot in a 2D plane, which we will later translate for our quadcopter, and whose velocity ‘u’ could be directly controlled and has a State ‘x’.

- *Go-to-waypoint:* In this behaviour robot’s task is to move towards the Goal position irrespective of the environmental condition, as switching conditions take care of environmental factors by changing behaviours. Considering robot’s current state be x and robot’s desired state i.e., state at way-point’s position be x_w, we could define a control or behaviour u_{GW} as :

$$u_{GW} = K_{GW}(x_w * x) \tag{3}$$

- *Avoid Obstacle:* In this behaviour robot’s task is to move away from the obstacle’s position. Considering x_o be obstacle’s state, then we could define u_{AO} as avoid obstacle behaviour as :

$$u_{AO} = K_{AO}(x - x_o) \tag{4}$$

- *Follow Obstacle Boundary:* In this behaviour robot’s task is to follow an obstacle wall in order to avoid it. We could assume that obstacle has a circular boundary whose radius is equal to the distance at which robot wants to avoid it. Then, we could easily define Follow Wall behaviour by rotating Avoid Obstacle behaviour by 90 degrees. The direction of rotation, clockwise or counterclockwise, will be decided by Switching Conditions.

- Clockwise:

$$u_{FW}^{c} = \alpha R(-\pi/2)u_{AO} = \alpha \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} u_{AO} \tag{5}$$

- Counter-Clockwise:

$$u_{FW}^{cc} = \alpha R(\pi/2)u_{AO} = \alpha \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} u_{AO} \tag{6}$$

where α is a gain matrix and R(θ) is a Rotation Matrix, defined as :

$$R(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{7}$$

B. Switching Conditions

Consider the robot initially following Go-to-waypoint behaviour, minimum obstacle distance Δ. Now, when the robot reaches Δ distance near the obstacle, it has to choose other behaviour in order to respond to the environmental changes caused due to the obstacle. Following are the Switching Conditions:

- From Go-To-Waypoint to Follow Wall: Here Robot has two options, either it could follow u^c_{FW} or u^{cc}_{FW} depending upon its Dot product with u_{GW}. Considering Dot product of vector *v* and *w* denoted as:

$$[(v, w) = v^T w = |v||w|cos(\angle v, w)]$$

- Follow Wall- Clockwise:

$$|| \ x - x_o \ || = \Delta \tag{8}$$

$$(u_{GW}, u^c_{FW}) > 0 \tag{9}$$

- Follow Wall- Counter Clockwise:

$$|| \ x - x_o \ || = \Delta \tag{10}$$

$$(u_{GW}, u^{cc}_{FW}) > 0 \tag{11}$$

- From Follow-Wall to Go-To-Waypoint:
- Progress : Consider *τ* be the last time of switch and *x*(*τ*) denotes the state at the time *τ*.

$$|| \ x - x_w \ || < || \ x(\tau) - x_w \ || \tag{12}$$

- Clear Shot: This condition takes care of the fact that there is no other obstacle between the robot's current state and goal position.

$$(u_{AO}, u_{GW}) > 0 \tag{13}$$

5. IMPLEMENTATION

We have formulated our algorithm for a Point robot moving in a 2-D plane but we are using a Quad-copter that will navigate in the 3-D plane. To implement this Planner on our robot, we need to modify the algorithm such that we could get Linear Velocity *v*, Vertical velocity v_{up}, v_x & v_y in Quadvelocity frame to navigate quadcopter and Yaw value as an output of planner.

A. Altitude Hold

Quadcopter in Gazebo Simulator is equipped with a downward facing Sonar Sensor which publishes the quadcopter's height on *"*{*sonar down"* ROS topic. Considering current altitude be represented by *Z* and our desired height be $Z_{setpoint}$. Thus, altitude hold controller could be easily defined as :

$$alt\_error = Z_{setpoint} - Z \tag{14}$$

$$v_{up} = PID(alt\_error) \tag{15}$$

It will run in an outer-loop over other controllers and always maintain the quadcopter in a 2-D plane.

B. Translating Reference Signal to Robot's Model

Point robot model behaviours will be able to generate $u_{GW,}$ $u_{AO,}$ u^{c}_{FW}, u^{cc}_{FW}. Taking all of these generated control signals as u, which will be a 2x1 matrix :

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{16}$$

Heading angle, ⍰$_d$ calculated from this control signal :

$$\phi_d = atan(\frac{u_2}{u_1}) \tag{17}$$

We have used a pre-defined linear velocity, v. We could also get it from control signal:

$$v = \sqrt{u_1^2 + u_2^2} \tag{18}$$

We can get X-Y components of this velocity in Quad-velocity frame using transformation matrix [II-B], thus, v_x and v_y is :

$$v_x = -v * sin(\phi) \tag{19}$$

$$v_y = v * cos(\phi) \tag{20}$$

For Angular velocity w, we could use PID regulator with ⍰$_d$

$$e = \phi_d - \phi \tag{21}$$

$$w = PID(e) \tag{22}$$

where *PID* refers as

$$PID(e) = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \dot{e}(t) \tag{23}$$

6. RESULTS

A. Altitude Hold

PID values used for Altitude controller in simulations:

$K_P = 0.7$; $K_I = 0.005$; $K_D = 0.03$;

Setpoint = 10m

Response curve for holding altitude of 10m and then landing back is shown in figure 5.

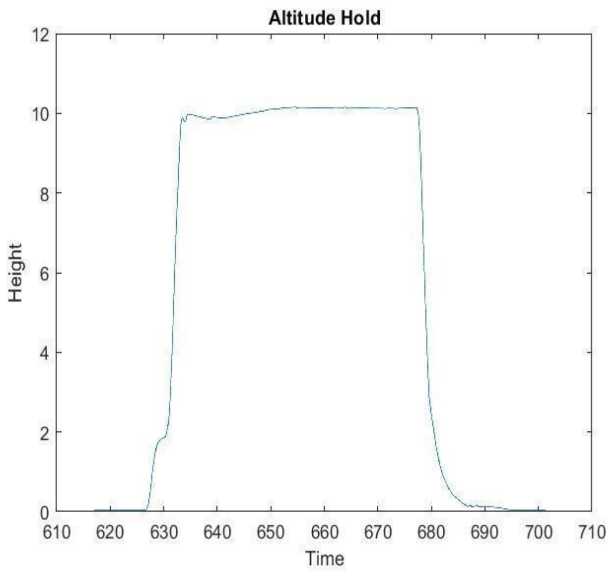


Figure 5: Altitude Hold.

B. Obstacle Avoidance Trajectory

- PID values used for controlling angle ⍰ :

  $K_P = 3.5$; $K_I = 0.05$; $K_D = 2.5$

- Minimum Distance, $\Delta = 1.2$m
- Linear velocity, $v = 0.5$m/s
- Altitude Setpoint = 1.3m
- First Way-point = (-8,0,1.3)
- Second Way-point = (8,0,1.3)

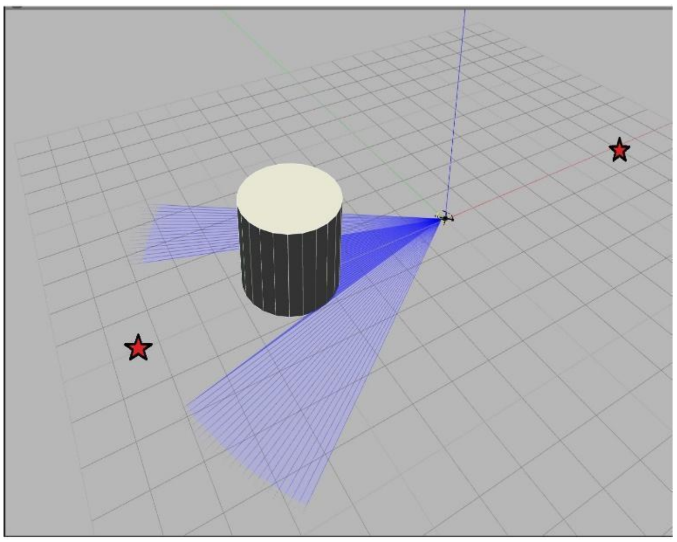


Figure 6: Simulation Scenario- way-Points has been shown by Stars.

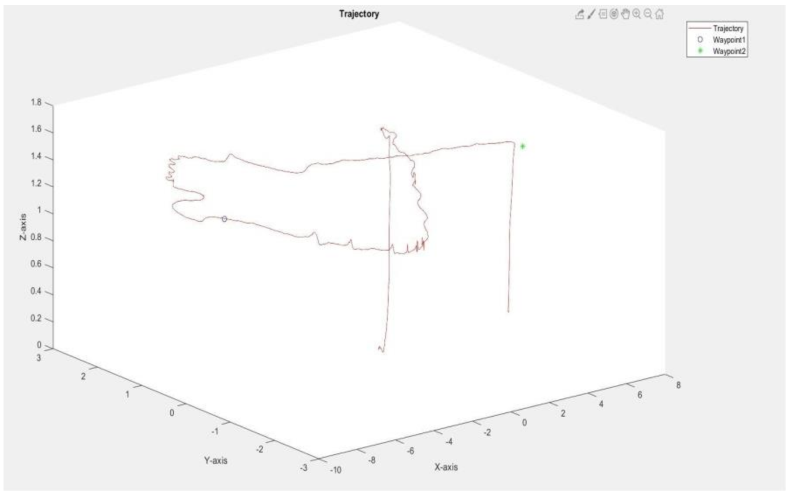


Figure 7: Trajectory Traced.

7. CONCLUSIONS

In this paper, we have developed a behaviour-based local planner for obstacle avoidance using Xbox Kinect as a vision sensor and ROS-Gazebo interface for simulation. We have provided pre-defined initial waypoints to the local planner. However, we could adjunct a global planner and feed our planner the trajectory generated from it. As future work, we plan to remove our quadcopter limit to a 2D plane for navigation. We will implement RGB-D SLAM[12] to generate a 3D map for a better understanding of the environment and use this map with MoveIT–ROS toolbox, for 3D collision-free trajectory.

REFERENCES

1. *Hrabar, S., "3D path planning and stereo-based obstacle avoidance for rotorcraft UAVs", IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 807–814, Sept 2008.*
2. *B.A.Kumar and D.Ghose, "Radar-Assisted Collision Avoidance/Guidance Strategy for Planar Flight", IEEE Transactions on Aerospace and Electronic Systems, vol. 37, no. 1, January 2001.*
3. *Y.K.Kwag and J.W.Kang, "Obstacle Awareness and Collision Avoidance Radar Sensor System for Low-Altitude Flying Smart UAV", Digital Avionics Systems Conference, October 2004.*
4. *Bachrach, R. H. and Roy, N., "Autonomous flight in unknown indoor environments", International Journal of Micro Air Vehicles, 2009.*
5. *J. L. Barron, D. J. F. and Beauchemin, S. S., "Performance of optical flow techniques."*
6. *R. Bernier, M. B. and Poitevin, P., "Dsa radar - development report."*
7. *Eric, N. and Jang, J., "Kinect depth sensor for computer vision applications in autonomous vehicles."*
8. *Arkin, R. C., Behavior-based Robotics. MIT Press, 1998.*
9. *Furrer, F., Burri, M., Achtelik, M., and Siegwart, R., RotorS— A Modular Gazebo MAV Simulator Framework. Cham: Springer International Publishing, 2016, pp. 595–625. [Online]. Available: https://doi.org/10.1007/978-3-319-26054-9 23*
10. *P. Bao, L. Z. and Wu, X., "Canny edge detection enhancement by scale multiplication."*

11. *ROS sensor msgs{pointcloud2 datatype. [Online]. Available: http://docs. ros.org/en/api/sensor msgs/html/msg/PointCloud2.html*

12. *J. Sturm, F. E. W. B. and Cremers, D., "a benchmark for the evaluation of rgb-d slam systems."*

13. *Park, Yeong-Sang, and Youngsam Lee. "Fast and Kinematic Constraint-Satisfying Path Planning With Obstacle Avoidance." International Journal of Electronics and Communication Engineering (IJECE) 5.3 (2016): 17 28 (2016).*

14. *Agarwal, Nisha, and Priyanka Yadav. "Reduce Energy Consumption in Wi-Fi Mac Layer Transmitter & Receiver by Using Extended VHDL Modeling." International Journal of Electrical and Electronics Engineering (IJEEE) 5.4 (2016): 33 42.*

15. *Mariappan, Er Dr Muralindran, Vigneswaran Ramu, and Thayabaren Ganesan. "Fuzzy Logic Based Navigation Safety System for a Remote Controlled Orthopaedic Robot (OTOROB)." Research and Development (IJRRD) 1.1 (2011): 21-41.*

16. *International Journal of Mechanical and Production Engineering Research and Development (IJMPERD) ISSN (P): 2249-6890; ISSN (E): 2249-8001 Vol. 8, Special Issue 7, Oct 2018, 1342-1347*